A delay-based routing metric

Baptiste Jonglez
École Normale Supérieure de Lyon

Matthieu Boutier
Université Paris-Diderot

Juliusz Chroboczek
Université Paris-Diderot

***Abstract*—In overlay networks, both local and long-distance links appear as a single hop to a routing protocol, and traditional routing metrics (based on hop count or packet loss) fail to take the differences between such links into account. In this paper, we describe the design and implementation of a metric based on packet delay that is designed to improve routing in overlay networks.**

Using delay naively leads to persistent routing oscillations, but the Babel routing protocol within which our metric is implemented employs a number of features to bound the frequency of oscillations and limit their impact by ensuring consistency even during reconvergence. We show experimental data that indicate that the protocol causes no oscillations in real-world situations, and has oscillations with a period on the order of minutes in artificially constructed topologies.

I. Introduction

An *overlay network* is a network created on top of an existing network. In more technical terms, an overlay network is a network the links of which are realised as flows (or connections) of the underlying network.

Overlay networks have many applications. They can be used as a transition technology, when the desired physical network does not exist yet — the transition to IPv6 was bootstrapped by running IPv6 within the *6bone*, an overlay over the existing IPv4 Internet. *Virtual Private Networks* (VPN) are a technology that allows a network node to appear connected at a place different from what is implied by the physical network topology, typically in order to work around topology-based security policies; *onion routing* [16] generalises this idea to large public virtual networks that are used to provide a modicum of anonymity to their users. Finally, by rerouting around failures faster than the underlying network does, overlay networks are used to improve the reliability of large-scale distributed systems in the presence of partial network failures. It is this last application that concerns us here.

A. Overlay networks for reliability

BGP, the routing protocol used in the Internet core, is designed to scale to very large networks. This implies a number of trade-offs, most notably relatively slow reconvergence after a network failure, on the order of minutes. Measurements indicate that at any one time a few percent of the expected routes are not available [14]. This implies that in a sufficiently large distributed system implemented on the Internet, such as a distributed cloud, at any time at least some of the participants will not be able to communicate.

There are multiple ways of dealing with this issue. An interesting approach is to design application algorithms that are able to deal with temporary failures; for example, the SMTP protocol used for electronic mail has a complex system of timeouts, retries and fallback servers that allows it to deal with temporary failures. A more recent example is that of the Kademlia distributed hashtable algorithm (used notably for locating peers in large-scale peer-to-peer file transfer applications), which is highly redundant in order to deal with arbitrary communication failures.

A more modular approach consists in delegating the reliability requirements to a lower layer. In this approach, the application blindly sends its data to the desired destination, and a lower layer uses an overlay network to route the data to the destination, using a routing algorithm with fast rerouting properties and with its own routing policies, possibly different from the policies used by the underlying network. This overlay network and routing algorithm can be implemented within the application layer (as an *ad hoc* library), as in *Resilient Overlay Networks* [2], which makes it possible to fine-tune the routing heuristics in an application-specific manner (e.g. prefer lower latency or higher reliability) without the need for cross-layer interactions. Alternatively, the overlay network can be implemented at the network layer, using familiar packet-switching technology, which reduces flexibility somewhat but allows using unmodified applications over the overlay.

B. Routing in a distributed cloud

SlapOS is a framework for building distributed cloud applications. SlapOS was initially implemented over native IPv6, which was found to be too unreliable. SlapOS was then modified to use a dense network (but not a full mesh) of virtual links [3], and route over it by using the off-the-shelf protocol Babel [4] with the hop-count metric.

This solution worked fairly well as long as the cloud was mostly local. Unfortunately, as soon as distant nodes were added, Babel started making routing choices that, while consistent with the shortest-hop metric, were clearly sub-optimal. Consider for example the topology in Figure 1, which consists of four nodes configured in an almost complete mesh. As long as all the links are operational, the shortest-hop metric yields optimal results — traffic local to Europe remains in Europe. However, if the link between Lille and Marseilles breaks, the shortest-hop metric does not allow the routing protocol to distinguish between the local route through Paris and the remote route through Tokyo, which is therefore chosen in roughly one half of the cases.

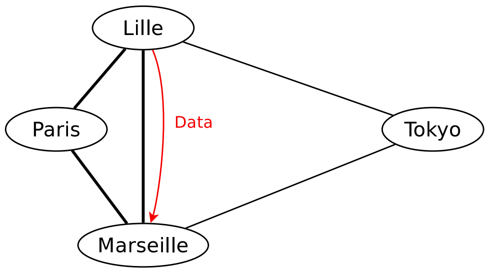

Fig. 1. A real-world topology

The shortest-hop metric is not precise enough for the distributed cloud. In this paper, we describe our work on extending the Babel routing protocol with a metric based on packet delay.

C. A delay-based metric

Our goal in this work is to extend the Babel routing protocol with the simplest possible metric that does reliably distinguish between local and non-local routes in an overlay network such as the one generated by SlapOS. Our metric is not meant to be the end-all of all metrics for overlay networks; still, the requirements of the application dictate a number of properties that it must have.

First, as one of the goals of the distributed cloud is to reduce cost, the metric must not require any manual configuration, which rules out manually configuring links as "local" or "remote". We have chosen to base our network on the *round-trip time* (RTT), or two-way delay, which is easily measured with off-the-shelf hardware with an accuracy sufficient to distinguish between Paris and Tokyo. (One-way delay might lead to a more generally useful metric in the presence of asymmetric network congestion, but it is more difficult to measure and is not required for this particular application.)

Second, the algorithm must be easy to implement on cheap off-the-shelf hardware, and, in particular, it must not rely on globally synchronised clocks. Since the links used in a distributed cloud are of varying quality, it must consume a negligible amount of additional network resources. Additionally, since the hardware used in the distributed cloud can be fairly loaded, it should be asynchronous, i.e. not require real-time response to query packets.

Finally, since delay can be caused by network congestion, using delay in a routing metric causes a feedback loop, which can cause persistent oscillations. We require that our algorithm provide reasonable stability, with a bound on the period of oscillations of at least a few minutes.

D. Stability issues

It is a widely believed piece of folklore in the routing community that using delay within a routing metric "doesn't work". Indeed, in a network subject to congestion, delay gives rise to a negative feedback loop: low RTT encourages traffic, which in turn causes the RTT to increase. In a discrete domain, such a feedback loop can cause persistent oscillations.

Consider for example the topology in Figure 2, where the links $A \cdot B$ and $A \cdot C$ are subject to congestion. Suppose that there is a significant amount of traffic from A to D. The routing protocol initially chooses some route, say the route through B; as the link $A \cdot B$ becomes congested, its RTT rises, so the routing protocol reroutes through C. The situation then reverses: the link $A \cdot C$ becomes congested, the protocol reroutes through B, etc.

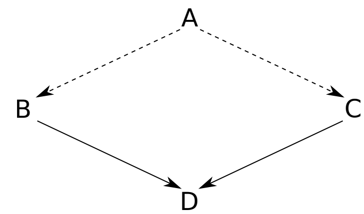

Fig. 2. A topology that causes oscillations

In the general case, such oscillations are unavoidable in the presence of congestion. However, the Babel routing protocol is somewhat less impacted by oscillations than some other routing protocols — Babel guarantees that the forwarding remains free of loops even during reconvergence, so while oscillations may cause packet reordering, they will not usually cause packet loss. What is more, our protocol extensions include two mechanisms, saturation and hysteresis, that cooperate to limit the frequency of oscillations: in Section IV-B, we provide empirical data that shows that in the classical oscillation-prone topology, the period of the oscillations is on the order of minutes, which is more than acceptable for our use-case.

E. Main contributions

Our extensions to the Babel routing protocol consist of four related techniques: an efficient, asynchronous algorithm for measuring RTT (Section III-A), a smoothing algorithm that eliminates short-term jitter (Section III-B), a saturation function that maps delay to a metric, and a hysteresis algorithm that is used in route selection (Section III-E). The former three techniques are derived from previous algorithms, while the hysteresis algorithm is new.

The main contribution of this paper consists in combining these techniques with the routing protocol Babel [4], and showing that our extended implementation is able to achieve a reasonably high level of stability. The implementation is publicly available, and is currently being deployed in production.

II. Related work

A. Use of RTT in routing protocols

In 1983, Mills described the use of RTT for routing in the DCNet [11], but didn't provide an evaluation of his protocol; the asynchronous algorithm that we use to measure RTT (Section III-A) is inspired by Mills' algorithm, which later became the basis for NTP [12]. A few years later, the "revised" routing protocol for the Arpanet [9] used a metric based on RTT in order to mitigate the congestion of the network;

stability issues were considered, and solved by saturating the metric, similarly to what we do.

Using a delay-based metric for routing has apparently been abandoned since then: to the best of our knowledge, no modern network has been using this method in recent years. Our interpretation is that congestion seldom occurs within the core of the network nowadays, and has moved to the edge, where there is little opportunity for routing optimisations: congestion occurs in the "Customer Premises Equipment" (e.g. the ADSL modem) which cannot be routed around.

The proprietary routing protocols IGRP and EIGRP [1] use a parameter called "delay" for computing their metric. However, this value is statically configured by the operator rather than determined dynamically, and this feature is therefore unrelated to the techniques considered in this paper.

B. Overlay networks

Overlay networks are an old idea, and there is a wide range of litterature describing their various applications. In this paper, we are concerned with the use of overlay networks to increase reliability, as described in Detour [14].

The techniques most similar to ours are the ones used by *Resilient Overlay Networks* (RON) [2], where the authors build an overlay network to increase reliability and use a variety of metrics, controlled by the application, to perform routing. Unlike our work, however, RON is layered above UDP and performs routing within the application layer: this makes implementation simpler and makes it easier to provide multiple routing metrics, but requires changing all applications to link with the RON library and use its primitives for communication. In contrast to RON, our network-layer approach allows the use of unmodified applications and is completely oblivious to the transport-layer protocol being used.

III. RTT-BASED ROUTING

In this section, we describe the issues related to integrating an RTT-based metric in the Babel routing protocol.

A. Measuring RTT asynchronously

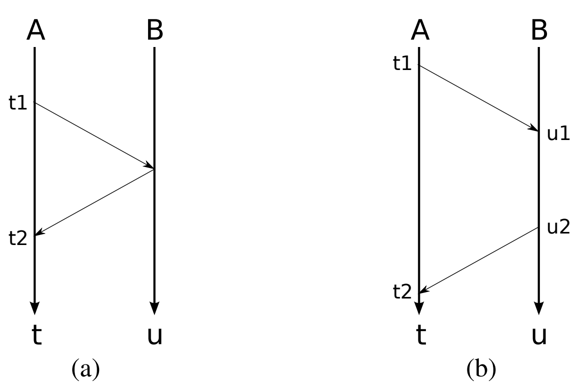

Fig. 3. RTT measurement

The simplest way to measure RTT between nodes A and B (Figure 3(a)), as performed e.g. by the *ping* program, is to send a single "echo request" packet from A to B, and have B immediately respond with an "echo reply". This is a simple and intuitive algorithm that does not require synchronised clocks; unfortunately, it requires a synchronous reply from B, which is not necessarily easy to integrate within an existing routing protocol.

Like most modern routing protocols, Babel has a fairly sophisticated scheme for scheduling outgoing messages. Roughly speaking, messages are delayed by a random time (at most one half of the *Hello* interval) in order to avoid global synchronisation [6] and to make it possible to aggregate multiple messages into a single packet. Adding synchronous messages to Babel would require a moderate amount of changes to the protocol, increase the amount of network traffic that it generates, and might cause unexpected issues with node synchronisation.

Fortunately, the problem of measuring RTT asynchronously has been solved before, and was used by Mills in his *HELLO* routing protocol [11] and in the NTP clock synchronisation protocol [12]. In Mills' algorithm (Figure 3(b)), a node A sends a packet p_1 with its local timestamp; B saves p_1's reception timestamp u_1 according to its local clock. At some later time, a node B sends a packet p_2 with a copy t_1 of p_1's timestamp, its timestamp u_1, and the timestamp u_2 of p_2. When node A receives the packet p_2 at local time t_2, it computes the difference

$$(t_2 - t_1) - (u_2 - u_1)$$

which yields the RTT. Note that each of the terms in this difference uses a single clock — hence, no clock synchronisation is necessary. Except for the first packet, all packets exchanged in Mills' algorithm carry three timestamps: therefore, each node computes a new RTT sample for each received packet, which is twice as efficient as the naive *ping* algorithm.

A further refinement is possible. On a multi-access network, a packet's timestamp is valid for all neighbours; it is only the echoed timestamps which must be sent to a particular peer. In Babel, we attach a timestamp to each *Hello* message, which is sent over multicast to all neighbours. The echoed timestamp is piggybacked to *IHU* ("I Heard You") messages, used for reverse reachability detection, which are conceptually unicast (but usually sent over multicast). In order to make it possible to perform Mills' computation, we ensure that every *IHU* is accompanied with a *Hello* in the same packet. Therefore, the cost of implementing Mills' algorithm is just a few octets per *Hello* and *IHU* message, with no additional packets sent.

B. Smoothing

The RTT samples obtained by the algorithm described above contain a varying amount of *jitter*, or short-term noise. Figure 4 shows the samples obtained over a period of almost one hour over a GRE tunnel between Paris and Tokyo, at a time when the RTT was particularly stable. Before time 1300, the samples are roughly constant, with a single outlier. At time 1350, something happens (rerouting?), there are a few outliers, after which time the samples are roughly constant again, with a small number of outliers.

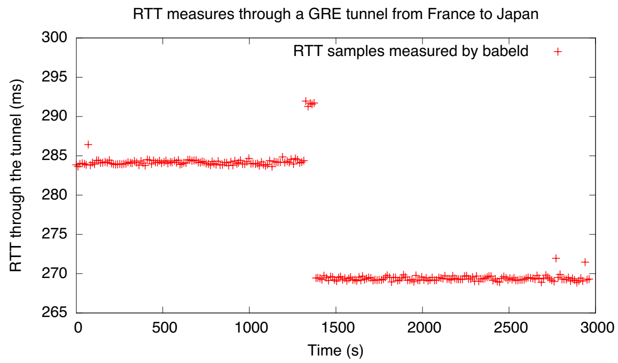

Fig. 4. RTT through a tunnel from Paris to Tokyo

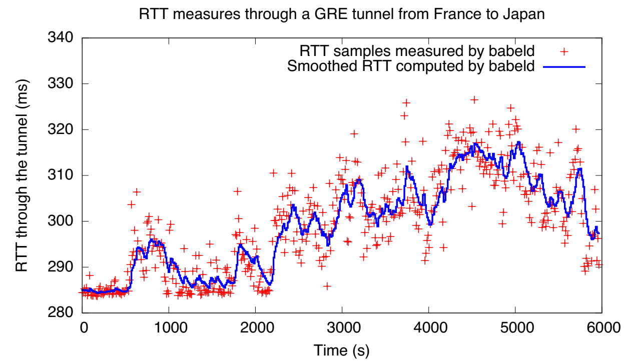

Fig. 6. Effect of smoothing on an unstable RTT

Obviously, we are interested in the medium-term latency averages (285 ms before time 1500, 270 ms after that), rather than in the random jitter. For that reason, we smooth the RTT data using an exponential average analogous to the one used by TCP [13]. More precisely, for every new RTT sample RTT_n, our RTT estimate RTT is updated as follows:

$$\mathrm{RTT} := \alpha \cdot \mathrm{RTT} + (1 - \alpha) \cdot \mathrm{RTT}_n$$

The value α is currently set to 0.836 by default (which is consistent with TCP's recommendation of 0.8 to 0.9). The results of this smoothing are shown in Figure 5.

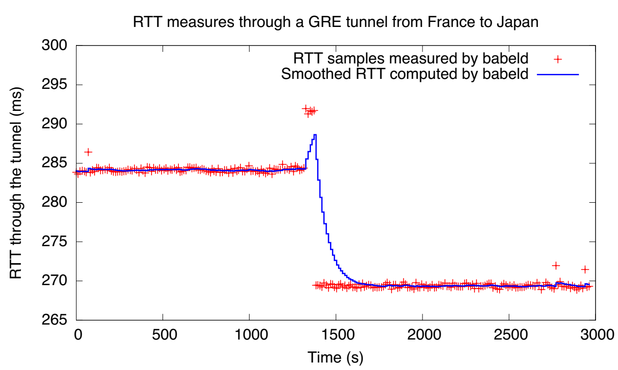

Fig. 5. Effect of smoothing on RTT

Figure 6 shows the behaviour of the same tunnel at a different time, when the RTT exhibited much larger variation. While the raw data is much more chaotic, the smoothing algorithm is able to provide useful data.

C. Accuracy and clock skew

As noted above, Mills' algorithm does not require synchronised clocks. However, its accuracy is limited by two factors. First, packets must be timestamped just before they are sent and just after they are received: if sent packets are timestamped too early, or received packets too late, the RTT will be overestimated. Second, the two clocks must progress at roughly the same rate: if one clock is significantly faster than the other, RTTs will be overestimated on the fast side and underestimated on the slow one.

Concerning the first issue, we have put some care into ensuring that timestamps are generated in a timely manner. Babel's packet formatter formats *Hello* messages with zero timestamps; the timestamps are filled in just prior to emission. On the receiving side, however, timestamps are only parsed after packet validation. Our tests on a local gigabit Ethernet indicate that we overestimate RTT by 0.4 ms as compared to the *ping6* command, and introduce a moderate amount of jitter, on the order of 0.1 ms. This is acceptable for the intended application.

As to the second issue, a clock skew of δ introduces a maximum error of $\delta \cdot \tau$, where τ is the maximum interval between two IHU messages (12 s by default). Typical computer clocks have clock skew on the order of 10 ppm, which should yield an error of at most 0.1 ms. Interestingly, our tests indicate that clock skew increases dramatically when one peer enters a power-saving mode: in that case, we have witnessed asymmetric errors of more than 1 ms, an order of magnitude more than the expected value. Even these extreme values, however, are within the accuracy required for the intended application of our protocol.

D. An RTT-based metric

In the previous section, we described how to measure RTT precisely and cheaply. The RTT alone, however, does not directly constitute a metric: we need to somehow map RTT values to an additive metric.

As far as the Babel routing protocol is concerned, a metric is just a 16 bit integer. While it would be possible to map RTT to a metric proportionally (just multiplying it by some suitable constant), this would favour low-RTT links too much, and prefer multiple low-RTT hops to a single moderate-RTT hop. What is more, it would yield arbitrarily large metrics for large RTT links, which, as we shall see in Section IV-B, has a negative effect on stability.

Instead, we map RTT to metrics using the piecewise affine function described in Figure 7. For RTTs below a value `min-rtt` (10 ms by default), a link is considered "good", and its metric is the fixed value `min-cost`. For RTTs above `max-rtt` (120 ms by default), the link is "bad", and its cost is the fixed value `max-cost`. For intermediate RTTs

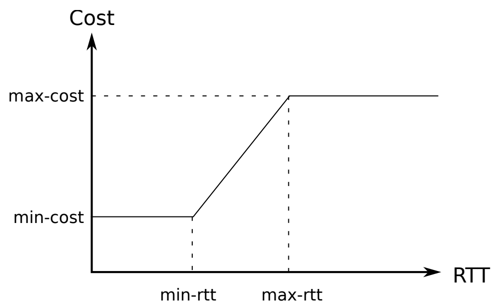

Fig. 7. Deriving cost from RTT

between `min-rtt` and `max-rtt`, the resulting cost is an affine function of the RTT.

This mapping has two essential properties. First, all link metrics are no smaller than `min-cost`, which guarantees that even very low RTT links are not seen as "free" — in a very low latency network, our metric degenerates to the shortest-hop metric. Second, all high-RTT links are treated equally, which, as we shall see in Section IV-B, is essential in order to limit the frequency of route oscillations in congested networks.

E. Hysteresis

In traditional routing protocols, metrics tend to vary discontinuously, by discrete amounts. Hence, a traditional routing protocol can afford to switch routes as soon as a route's metric becomes lower than that of the currently selected one. When continuous metrics are used that measure real-world parameters, this is no longer the case: the metrics of two routes could oscillate around a similar value, leading to frequent route oscillation. For that reason, the Babel routing protocol applies a hysteresis algorithm to the metrics that it considers for route selection. As we shall see in Section IV-B, this hysteresis is essential to the stability of delay-based routing.

The algorithm is as follows. For every route, Babel maintains two metrics: the *advertised metric* M_a, which is obtained from neighbours and readvertised to other nodes, and the *smoothed metric* M_s. The smoothed metric is initialised to the advertised metric, and is periodically updated according to the formula:

$$M_s := \beta(\delta) \cdot M_s + (1 - \beta(\delta)) \cdot M_a$$

where δ is the delay since the last update of M_s, and $\beta(\delta)$ is a value chosen so that M_s converges towards M_a exponentially with a time constant of 4 s (in base 2). Intuitively, M is an estimate of how good the route is right now, while M_s is an estimate of how good the route has been recently.

Babel's route selection algorithm works as follows. When the advertised metric of the current route becomes infinite (the current route has been retracted), the algorithm chooses among the routes with finite advertised metric the one that has the smallest smoothed metric: when a route is lost, it immediately picks the route with the best recent history. When the current route has a finite advertised metric (the current route is believed to be functional), the algorithm switches to some other route when both the latter's metrics are better than those of the currently selected route: it only switches to a route that is better both right now and in recent history. In effect, the algorithm reconverges immediately when the current route is lost, but otherwise delays switching to a better route until the new route has been shown to be stable for a few seconds.

The hysteresis algorithm may appear similar to the smoothing algorithm described above, but there are good reasons why these are separate. Babel is a modular protocol, and metric computation is separate from route selection. The smoothing algorithm is part of the metric calculation, and is designed to extract a smooth signal from the noisy RTT samples; it is specific to the RTT metric. The hysteresis algorithm, on the other hand, is part of the (metric-independent) route selection procedure, and its only purpose is to improve stability by delaying switching to a better route.

IV. Experimental evaluation

In this section, we show some empirical data describing the behaviour of our implementation of the algorithm described above.

A. Real-world behaviour

We have tested our implementation on a small overlay network deployed over the Internet, consisting of four nodes, three of which are in France and one in Japan. The topology of the overlay network is the one in Figure 1. Each node is running Linux, and the links are implemented using OpenVPN over UDP (without cryptography). All Babel instances are run with `rtt-min` equal to 10 ms, `rtt-max` equal to 200 ms, `min-cost` equal to 96 and `max-cost` equal to 246 Throughout the experiment, Lille is sending data to Marseilles.

Figure 8 shows the incoming throughput in Marseilles over each of the local interfaces. Initially, all links are up, so the data arrives directly from Lille. Around minute 13, the direct link between Lille and Marseilles is shut down; after a few dozen seconds, the failure is detected, and the data is rerouted through Paris. Around minute 14, the Paris link is shut down, and the data is rerouted through Tokyo. Finally, after minute 15, the links are reestablished; when this is detected, the data is rerouted through the direct low-latency link.

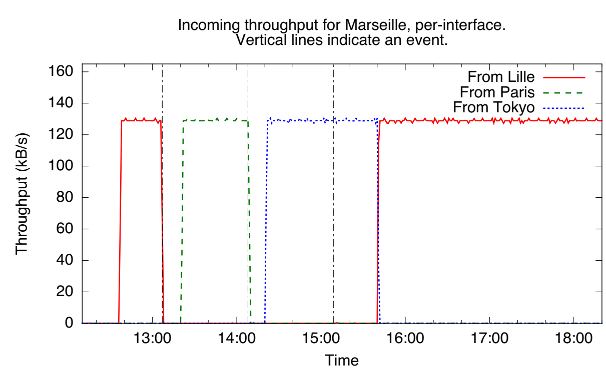

Fig. 8. Throughput in Marseilles

Figure 9 shows the metrics of the different routes during the experiment. It shows that the links remain uncongested: all of the metrics remain roughly constant throughout the experiment.

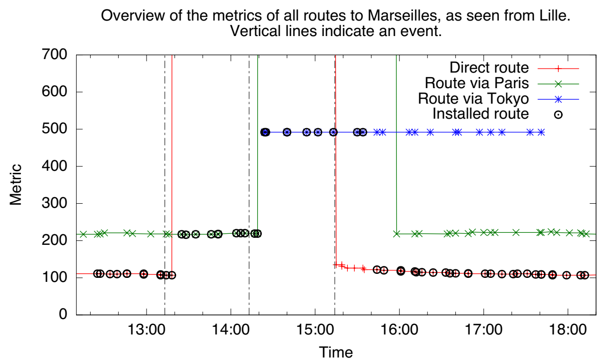

Fig. 9. Metrics in Lille

B. Simulated topology and oscillations

The previous experiment uses links of different natural latencies that remain uncongested throughout the experiment. We believe that this is representative of real-world conditions in overlay networks; however, since the traffic that we generate does not significantly impact the latencies of the links, the feedback loop described in Section I-D does not occur, and there are no stability issues.

In order to test our algorithm's stability properties in a situation that is prone to oscillations, we have simulated a network consisting of two exactly identical parallel routes that are subject to congestion. The topology is that of Figure 2; the links $A \cdot B$ and $A \cdot C$ have their throughput artificially limited, and are therefore subject to congestion, while the links $B \cdot D$ and $C \cdot D$ are uncongested.

As expected, routing in this somewhat pathological topology is subject to oscillations. Figure 10 shows the RTTs of the two congested links. The routing protocol chooses one of the two routes, the RTT of which subsequently increases; after a few minutes, the protocol reacts to the increase of the RTT and switches to the other route; the situation then repeats, *ad nauseam*. However, the frequency of the oscillations remains bounded, with a time constant of roughly 5 minutes.

Two mechanims collaborate to limit the frequency of oscillations. The saturation of the cost function ensures that both congested links spend part of their time in the saturated state. Hysteresis ensures that Babel doesn't switch routes as long as both metrics are saturated.

Figure 11 shows an experiment performed in the same topology, but with an unbounded cost function (both `rtt-max` and `cost-max` set to very high values, chosen so that the slope of the curve remains the same as in the previous experiment). The oscillations are now much faster (less than a minute), which shows the importance of a bounded cost function.

In this experiment, the congested links are the ones close to the sender, which ensures fast reaction to changing conditions.

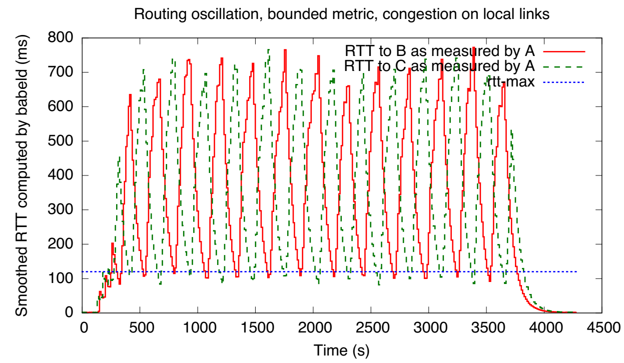

Fig. 10. RTT oscillation in a congested diamond

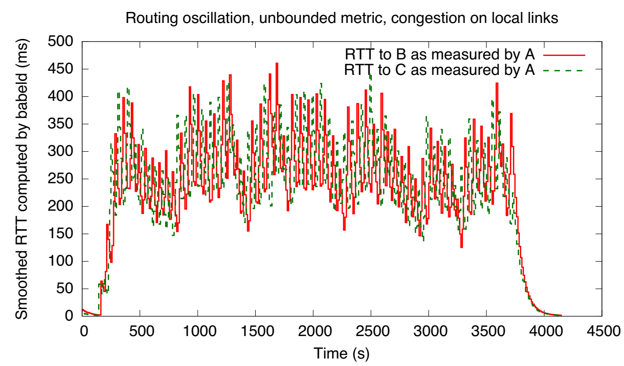

Fig. 11. RTT oscillation with saturation disabled

We have repeated the experiment with the links $B \cdot D$ and $C \cdot D$ being the ones subject to congestion; as expected, the behaviour is similar, but with slightly slower oscillation.

V. Conclusions and further work

In this paper, we have described a working implementation of a delay-based routing metric that is currently deployed in production. We have shown an algorithm that measures RTT while having a negligible impact on the amount of routing protocol traffic, and have shown techniques that mitigate the stability issues that are caused by using delay as input to the routing metric, and that are good enough to limit instability in the most hostile examples that we could construct.

While the functionality of our protocol is sufficient for the overlay networks that we consider, there is a number of related issues that still remain open.

a) One-way delay: The metric described in this paper is based on the round-trip time, or two-way delay. The congestion control community have repeatedly shown that one-way delay behaves better than two-way delay, at least as far as congestion control algorithms are concerned [10], [15], at the cost of much more complex algorithms. It would certainly be interesting to find out whether there are any real-world cases where one-way delay performs significantly better than RTT as a basis for a routing metric.

b) Arbitrary choices and theoretical study of stability: There are a number of arbitrary choices in our algorithm:

the constants used for smoothing and filtering, the amount of hysteresis applied, and, above all, the function used for mapping an RTT value to a metric. While we have empirically checked that these particular choices work well, at least for the particular application under consideration, there are almost certainly other choices that would work just as well and perhaps better. On a related note, these statically configured values could conceivably be determined dynamically by an improved algorithm.

More generally, we lack an in-depth theoretical understanding of the performance of our algorithm, in particular of its stability. There exist techniques for the theoretical study of the stability of distributed systems, and some of those would seem to apply to our case.

c) Other routing protocols: The metric described in this paper has been implemented within the loop-avoiding distance-vector protocol Babel. While our work is not in principle specific to Babel, Babel's characteristics make it particularly suitable to dynamically computed metrics: Babel avoids loops even during reconvergence, employs delayed updates, and supports fairly flexible route selection policies. However, the high period of the oscillations that we observed makes us optimistic that similar techniques could be applied to less tolerant routing protocols.

d) Other applications: After we initially published the code of our implementation, one researcher has expressed interest in studying its suitability for networks other than overlays. There is some support to the feeling that the metrics currently used in wireless mesh networks (such as ETX [5] or physical-layer metrics) are not satisfactory, because they are a poor predictor of network performance, because they are too slow to react to changing conditions, or because they are too difficult to implement. We hold some hope that, at least for some MAC layers, an accurate measurement of delay might be a good indicator of lower-layer congestion, and therefore could serve as one component of a metric for wireless mesh networks.

ACKNOWLEDGMENT

We are grateful to Julien Muchembled of Nexedi for providing access to the cloud nodes used in our experiments.

CODE AVAILABILITY

Our extension is included in the sample implementation of Babel, and is available on https://github.com/jech/babeld.

REFERENCES

[1] Bob Albrightson, J. j. Garcia-luna aceves, and Joanne Boyle. EIGRP — A Fast Routing Protocol Based On Distance Vectors. In *Proc. Networld/Interop*. 1994.

[2] David G. Andersen, Hari Balakrishnan, M. Frans Kaashoek, Robert Morris. Resilient Overlay Networks. Proc. 18th ACM SOSP, Banff, Canada, October 2001.

[3] Ulysse Beaugnon, "Building a resilient overlay network: Re6stnet", *Internship at Nexedi*, September 2012.

[4] Juliusz Chroboczek, *The Babel Routing Protocol*. RFC 6126, February 2011.

[5] D. De Couto, D. Aguayo, J. Bicket, and R. Morris, A high-throughput path metric for multi-hop wireless networks. Proc. MobiCom 2003, 2003.

[6] Sally Floyd and Van Jacobson. The synchronization of periodic routing messages, *IEEE/ACM Transactions on Networking*, **2**:2, pp. 122-136, April 1994.

[7] Baptiste Jonglez. *A delay-based metric for the Babel routing protocol*. Third year internship report, Université Paris-Diderot. 25 August 2013.

[8] Baptiste Jonglez and Juliusz Chroboczek, *Delay-based Metric Extension for the Babel Routing Protocol*. Internet draft, 2013. (work in progress)

[9] Atul Khanna and John Zinky. The Revised ARPANET Routing Metric. *SIGCOMM'89 Symposium proceedings on Communications architectures and protocols*, pp. 45-56. ACM New York, NY, USA, 1989.

[10] Aleksandar Kuzmanovic and Edward W. Knightly, "TCP-LP: low-priority service via end-point congestion control", *IEEE/ACM Transactions on Networking*, **14**:4, pp. 739-752, August 2006.

[11] David L. Mills, *DCN Local-Networks Protocols*. RFC 891, 1983.

[12] David L. Mills, J. Martin, Ed., J. Burbank and W. Kasch, *Network Time Protocol Version 4: Protocol and Algorithms Specification*. RFC 5905, June 2010.

[13] John Postel. *Transmission Control Protocol*. RFC 793, September 1981.

[14] Stefan Savage, Tom Anderson, Amit Aggarwal, David Becker, Neal Cardwell, Andy Collins, Eric Hoffman, John Snell, Amin Vahdat, Geoff Voelker, and John Zahorjan. Detour: a Case for Informed Internet Routing and Transport. *IEEE Micro*, pp. 50-59, v 19, no 1, January 1999.

[15] S. Shalunov, G. Hazel, J. Iyengar and M. Kuehlewind, *Low Extra Delay Background Transport (LEDBAT)*. RFC 6817, December 2012.

[16] Paul F. Syverson, David M. Goldschlag, and Michael G. Reed. Anonymous Connections and Onion Routing. *Proceedings of the 18th Annual Symposium on Security and Privacy*, IEEE CS Press, Oakland, CA, May 1997.